Genatic Algorithm Based Architecture Search in Stacked LSTM Network for Time Series Forecasting

Linzhe Cai[1] , **Xinghuo Yu**[1] , **Chen Liu**[1] , **Chaojie Li**[2] , **Andrew Eberhard**[3]
[1]School of Engineering, RMIT University, Melbourne, 3000, VIC, Australia
[2]School of Electrical Engineering and Telecommunications, University of NSW, 2052, NSW, Australia
[3]School of Science, RMIT University, Melbourne, 3000, VIC, Australia
s3548838@student.rmit.edu.au; chen.liu3@rmit.edu.au; xinghuo.yu@rmit.edu.au

Abstract

Neural Architecture Search (NAS) has gained significant attention in recent years for its potential to automate the design of deep learning models. Although NAS has been widely studied in classification tasks, its application in regression tasks, such as time series forecasting, remains relatively unexplored. A genetic algorithm-based search method is proposed to expand traditional architecture search space by considering the number of neurons in different layers independently. Modified crossover and mutation operations provide a balance between search space and search efficiency. Four real-world open-source datasets in different fields are used to verify the effectiveness of the proposed algorithm. Some effective patterns are obtained and can be generalized to time series forecasting in different fields. Scatter diagrams are used to compare the performance of different types of patterns and illustrate the improvement from both accuracy (MSE up to 19.39 times smaller) and elapsed time (reduce at most by 62.97%).

1 Introduction

Deep learning's rapid progress has facilitated the creation of increasingly intricate neural network architectures, achieving unparalleled results across a range of tasks [Goodfellow *et al.*, 2016]. However, the design of these architectures traditionally involves a laborious manual process, demanding substantial expertise and experience. Neural Architecture Search (NAS), aiming to automate this process, employs optimization algorithms to discover the most effective network architectures for a specific task. While NAS has garnered considerable interest recently, demonstrating promising results in various classification tasks such as image recognition [Pham *et al.*, 2018], its application in regression tasks, including time series forecasting, remains largely untapped.

Time series forecasting [Chatfield, 2000] is an essential element in numerous real-world applications, encompassing stock price prediction, electricity price forecasting, and demand forecasting. Conventional methods like ARIMA models [Ahmed, 1979] have been extensively employed for these tasks. Javier [Contreras *et al.*, 2003] used the statistic-based method to predict the price of electricity, while Conejo [Conejo *et al.*, 2005] combined ARIMA with wavelet transform to improve the performance, and pointed out a wavelet function of type Daubechies of order 5 and decomposition level 3. However, the emergence of deep learning techniques, such as long short-term memory (LSTM) networks [Hochreiter and Schmidhuber, 1997], has led to enhanced performance in many instances. Sreelekshmy [Selvin *et al.*, 2017] proposed a sliding windows model which can overcome the outliers that are hard to predict, while Arif [Sunny *et al.*, 2020] proposed a bi-direction LSTM model, which may solve the underfitting problem when training after necessary epochs.

With the development of big data, the time series datasets and their prediction become more and more complex, which presents the requirement of the application of multiple layers stacked LSTM. However, it is hard to define the appropriate hyper-parameter, especially under multiple-layer architecture. Although there has been some research related to neural architecture search for LSTM [Greff *et al.*, 2016; Gorgolis *et al.*, 2019], these studies are insufficient, because most of them just arbitrarily choose the same number of neuron in different layers, but did not discuss the necessity of so-called standard configuration. As a meta-heuristic optimization inspired by the process of natural selection and genetic inheritance, genetic algorithm (GA) was developed by John Holland [Holland, 1992] and has since become a widely used optimization technique in various fields.

This article is focused on a GA-based architecture search method, trying to figure out effective (more accuracy but less running time) architecture patterns for stacked LSTM engaging time series forecasting tasks. Our contributions are:

- Extend the search space of stacked LSTM architecture to different layer's own different numbers of neurons.
- Improve the search speed by GA and modified crossover and mutation operations.
- Obtain some effective LSTM architecture patterns for time series forecasting which are generally applicable.

The remainder of this paper is organized as follows: Section 2 reviews recent studies in NAS and LSTM, Section 3 gives background about stacked LSTM and GA, Section 4 elaborates our methodology of GA base LSTM architecture

search, Section 5 demonstrates the numerical experiments, and Section 6 gives conclusion.

2 Related Works

2.1 Neural Architecture Search

Neural Architecture Search (NAS) is a significant subset of AutoML, with considerable overlap with hyperparameter optimization and meta-learning. The traditional methodology of NAS, as categorized by Elsken [Elsken *et al.*, 2019] comprises the search space, search strategy, and performance estimation.

Search Space

The optimization problem's complexity is primarily dictated by the choice of the search space [Elsken *et al.*, 2019]. While a global search space provides the benefit of a high degree of freedom by considering all potential neural architectures, it also incurs a significant search cost [Ren *et al.*, 2021]. To alleviate this, various search spaces allow for the representation of architectures as fixed-length vectors, enhancing the compactness and reducing the complexity of NAS tasks. For instance, the search space in Zoph [Zoph *et al.*, 2018] is represented as a 40-dimensional vector.

Search Strategy

Exploring the space of neural architectures can be achieved through various search strategies. Early successes in NAS were marked by the use of Bayesian optimization since 2013 [Bergstra *et al.*, 2013]. A recurrent neural network (RNN) policy for the sequential sampling of a string that encodes the neural architecture is adopted by Zoph [Zoph and Le, 2016]. NAS is approached as a sequential decision-making process by Cai [Cai *et al.*, 2018]. GA have also been introduced by Miller [Miller *et al.*, 1989] for searching network architectures, while it extended to structure and weights by Stanley [Stanley and Miikkulainen, 2002].

Gradient-based methods for weights updating and evolutionary algorithms for neural architecture searching were later employed by Real [Real *et al.*, 2017]. A differentiable architecture search (DARTS) method, enabling the selection of candidate connections via softmax of all possible operations is introduced by Liu [Liu *et al.*, 2018]. I-DARTS is provided by [Jiang *et al.*, 2019] to circumvent the limitation of DARTS becoming a local model through further applied Softmax. Efficient Architecture Search (EAS), proposed by Ashok [Ashok *et al.*, 2017], which employed an encoder network as a meta-controller to learn the low-dimensional representation of existing neural architectures.

Performance Estimation

To accelerate performance estimation after obtaining the architecture, several strategies have been developed. Lower fidelity estimates, learning curve extrapolation, weight inheritance, or network morphisms are included. While the latter two are easily applicable during GA without increasing memory load [Elsken *et al.*, 2019; Real *et al.*, 2017]. In the context of gradient descent methods, one-shot learning with weight sharing has proven to be an effective speed-up method, reducing computation from thousands to a few GPU days [Pham *et al.*, 2018]. Another approach proposed by [Bender *et al.*, 2018] involved randomly sampling a plethora of candidate architectures from the hyper-network via parameter sharing and then evaluating these architectures to identify the best one. While NAS has been widely discussed recently, most of them focused on classification like image recognition instead of regression like time series forecasting.

2.2 Long Short Term Memory

As one of the most widely used deep learning models, LSTM is essentially a type of recurrent neural network (RNN) architecture, first proposed by Sepp and Juergen [Hochreiter and Schmidhuber, 1997]. The motivation is designed to handle the problem of vanishing gradients (gradients of the error function with respect to the parameters of the network become too small to train) in traditional RNNs.The LSTM model was further improved in 1999 by Felix Gers [Gers *et al.*, 2000] by introducing a forget gate, which allowed the model to discard some information when necessary, thereby enhancing LSTM's ability to process more complex data. A simplified LSTM variant called the Gated Recurrent Unit (GRU) is proposed by Kyunghyun Cho [Cho *et al.*, 2014], which merged the forget and input gates into a single "update gate" and combined the cell state and hidden state, greatly simplifying the model.

There are some studies about the neural architecture search of LSTM. The interaction between all pairs of hyperparameters of LSTM is concluded by Klaus [Greff *et al.*, 2016], revealing that these hyperparameters are almost mutually independent. A GA for neural hyperparameter optimization is proposed by Nikolaos [Gorgolis *et al.*, 2019], a group of standard configurations accepted by the community is mentioned as a standard configuration of medium size for RNN. However, both of them considered the same number of neurons in different hidden layers but did not demonstrate its effectiveness/necessity.

To fill the research gap mentioned above, we design a GA-based stacked LSTM architecture search for regression problems, namely time series forecasting. While searching for the number of neural network layers, the algorithm searches for the number of neurons in each layer simultaneously, classifying and comparing the best-performing alternative architectures. Although it dramatically increases the search space, our method can perfectly balance the performance and time complexity through modified crossover and mutation operations of GA.

3 Preliminaries

3.1 Stacked LSTM

In the LSTM network, each neuron comes equipped with a memory cell for preserving information over an extended period. This memory cell is regulated by three types of gates: input, output, and forget [Hochreiter and Schmidhuber, 1997]. The input gate manages the influx of fresh data into the memory cell. Simultaneously, the output gate oversees the outflow of information from the memory cell. Lastly, the forget gate determines the quantity of previously stored data that remains conserved in the memory cell. The architecture

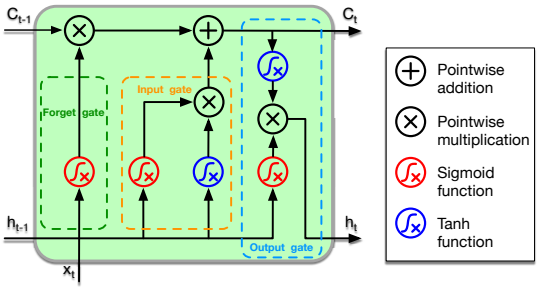

Figure 1: The architecture of a LSTM node.

of the LSTM node is given in Figure 1. The computation of each gate can be summarized as follows:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{1}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{2}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{3}$$

where x_t represent input at the current time step, and h_{t-1} represent previous hidden state. While W_{xi}, W_{xf}, W_{xo}, W_{hi}, W_{hf}, and W_{ho} are weight matrices that map inputs and hidden states to the gate units, b_i, b_f, and b_o are vectors that adjust the biases of the gate units. σ represents the Sigmoid function, and i_t, f_t, and o_t represent the output of the input gate, forget gate, and output gate respectively. Additionally, the computation of the hidden state (h_t) also involves the cell state (c_t) and the hyperbolic tangent function (tanh):

$$h_t = o_t * tanh(c_t) \tag{4}$$

By controlling the flow of information through the memory cell using these gates, LSTM networks are able to selectively retain important information over long periods of time while discarding irrelevant information. This makes them particularly well-suited to sequence learning tasks that require the network to remember important information over many time steps.

For complex sequence tasks, a single-layer LSTM might not have sufficient expressive power. To address this limitation, researchers have proposed the use of stacked LSTM, where multiple LSTM layers are stacked together to enhance the model's capacity.

The benefits of stacked LSTM and its ability to capture complex patterns and long-term dependencies in sequence data are discussed by Greff [Greff *et al.*, 2016]. Stacked LSTMs outperformed single-layer LSTMs in capturing hierarchical structures and long-term dependencies in sequences as demonstrated by Bai [Bai *et al.*, 2018]. The stacked LSTM model effectively captured the spatio-temporal dependencies in traffic data as concluded by Cui [Cui *et al.*, 2020], leading to accurate predictions.

The definition of stacked LSTM can be represented as the following equation:

$$h_t^l, c_t^l = \text{LSTM}(x_t, h_t^{l-1}, c_t^{l-1}), \quad l = 1, 2, \ldots, L \tag{5}$$

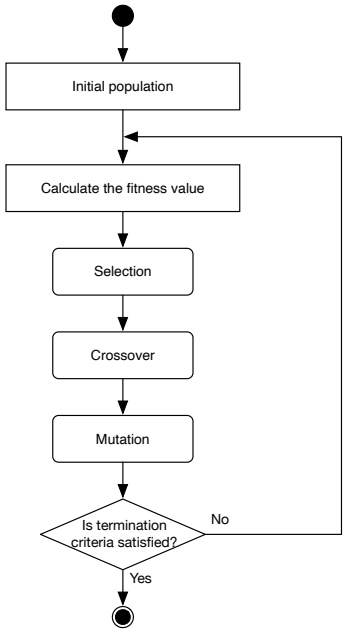

Figure 2: The flow chart of classic GA.

where h_t^l represents the hidden state of the l^{th} LSTM layer at time step t, c_t^l represents the cell state of the l^{th} LSTM layer at time step t, x_t represents the value of the input sequence at time step t, and L represents the number of LSTM layers. The input of the first LSTM layer is the input sequence x_t, and the output of the L^{th} LSTM layer is the final output.

Although stacked LSTM can enhance the model's expression and memory capability by increasing the number of LSTM layers, it should be noted that increasing the number of layers also increases the model's complexity and training difficulty, requiring appropriate adjustments and optimizations.

3.2 Genetic Algorithm

GA [Goldberg and Sastry, 2007] works by iterative searching for the optimal solution to a given problem. The algorithm starts with a randomly generated population of candidate solutions, each represented by a set of parameters or genes. The fitness of each candidate solution is evaluated based on a fitness function that measures its quality or performance to determine whether it should be included in the next generation. The fitter solutions are then selected to reproduce and create offspring through genetic operators such as crossover and mutation. This process is repeated for multiple generations until a satisfactory solution is found or a termination criterion is met. The flow chart of classic GA is given in 2.

In the realm of biological genetic encoding, there is an indirect correlation between the genotype and the phenotype. Typically, the phenotype comprises exponentially more structural elements than the number of genes present in the genotype [Gauci and Stanley, 2007]. As a result, identifying structures with a multitude of parts may necessitate a mapping that transforms a few dimensions into many. This process is often achieved through an indirect encoding method as suggested by Stanley [Stanley, 2007].

4 GA-based stacked LSTM NAS

According to Section 2, current studies didn't consider the possibility of different layers may own different numbers of neurons. Without the limitation of consistent neuron number, the search space increases dramatically, and there is a high probability of finding architecture with better performance. To improve the current study, our algorithm searches for the number of neurons in each layer independently. Regression problems, namely time series forecasting, are considered as the evaluation methods and the fittest architectures are tested and compared.

The framework we develop to determine the optimal stacked LSTM architecture for time series forecasting is illustrated in Figure 3. The first step is data preprocessing according to the quality and characteristics of the original datasets. Data cleaning for series with more than 10% lost data; feature selection for both input and output; one-hot encoding for non-value features; normalization for features with uncommon scale; and split datasets to training, validation, and testing for architecture searching and evaluation objects.

While extending the search space has a large likelyhood to obtain high-performance architecture, it increases the computation burden as discussed in Section 3. To overcome the computational burden of brute-force search [Heule and Kullmann, 2017], GA is considered as the solution. As the correlation ratio of the number of layers and the number of neurons in different layers is strong, searching those hyperparameters simultaneously is necessary and significant. An unfixed length genotype is considered to map LSTM architecture, in which the number of hidden layers (l) occupis the first gene of individuals, and the number of neurons in each layer (N) occupy the rest locations of the gene, which can be chosen independently.

$$l \in [1, 2, 3, ...l] \tag{6}$$

As we are more concerned with the effective pattern with a wide application range instead of finding the optimal architecture for a specific problem, we arbitrarily give a group of integers instead of giving a continuous integer range as the alternatives of neuron numbers, and the relationship between adjacent values is half and double.

$$N = \{N_1, N_2, N_3, ...N_l\} \in [\theta, 2\theta, 4\theta, 8\theta, ..., 2^n\theta] \tag{7}$$

while the depth of the network is decided by l in Equation 6, the width of which is decided by θ in Equation 7, and n in Equation 7 represents the amplitude of variation of numbers of neuron in different layers, which play the role of a controller to balance the search space and convergence rate.

Three basic operations of GA, namely selection, crossover, and mutation are used to construct the gene of new individuals (offspring) from their parents, these operations are modified according to our target, and some filters are considered to filter out invalid offspring.

Mean Squared Error (MSE) is used as the fitness function in a GA to optimize a model's performance, which measures the average squared difference between the predicted and actual values in the validation set:

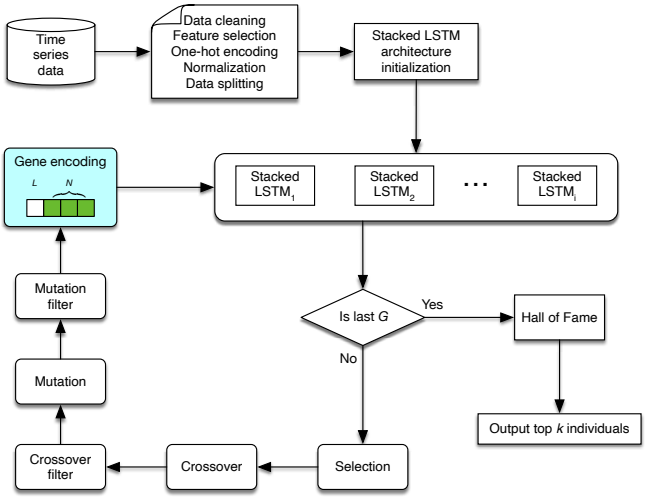

Figure 3: The framework of GA based LSTM architecture search

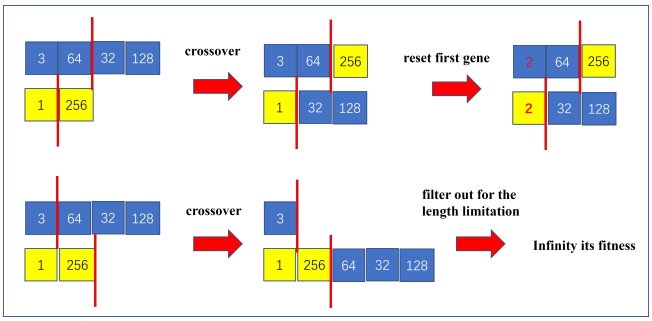

Figure 4: An example of GA messy-one-point crossover

$$Fitness = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{8}$$

where y_i and $\hat{y}_i$ represent the actual value and predicted value of the i^{th} sample respectively.

According to the fitness defined above, the smaller the better, the roulette method is inappropriate [Shukla *et al.*, 2015] as the selection operator. The tournament is considered, selecting a random subset of individuals from the population and then choosing the fittest individual from that subset, as the winner to breed the next generation.

As the length of gene encoding is uncertain, we consider a messy-one-point crossover instead of the normal one to ensure the diversity of offspring. Figure 4 illustrates the detail of the messy-one-point crossover operation and corresponding filter method. Two individuals (I) are taken as input, randomly select crossover points from parents respectively, and exchange the genetic material to the right of the crossover point between the two individuals. After that, reset the first gene according to the rest if necessary, as the top of Figure 4 shows. If the offspring can not satisfy the length range, as the bottom of Figure 4 shows. Filter it out through infinity its fitness value. Thus, it cannot be considered according to the tournament selection operator.

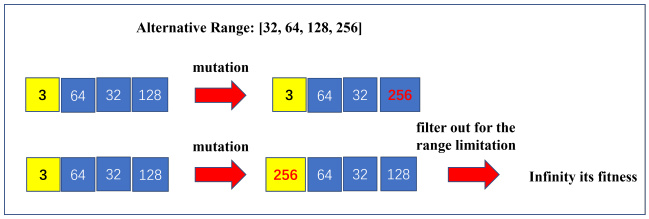

Figure 5: An example of GA customized mutation

Algorithm 1 GA based LSTM Candidate Architecture Search

```
Load dataset D with training sets
Prepare dataset X, Y ∈ D using window size w
Split X, Y into training and testing sets
Select number of generations G and number of individuals I
for each generation in G do
    Tournament Selection
    Messy-one-point Crossover
    Customized Integer Mutation
    Generate new offspring
    for each individuals in I do
        Create Stacked LSTM model M with architecture from individual I
        Train M using training set and hyperparameters H
        Evaluate M on validation set, calculate MSE
        Update Hall of Fame with best individuals (ranking)
    end for
end for
Output top k architectures A as well as their fitness value E
```

Customized integer mutation operation is considered with the corresponding filter method, and the detail is illustrated in Figure 5. Although we still change the value of an integer attribute in an individual with uniform probability, the replacement integer value chooses from a discrete range instead of a continuous one, which brings into correspondence with N_i in Equation 7, as the top of Figure 4 shows. Another filter is considered to prevent abnormal genes, namely first location mutation, from evolution, similarly through maximizing its fitness value.

Algorithm 1 implies the proposed methodology above.

For all time series datasets, we divide them into three parts: training dataset, validation dataset, and testing dataset. While X presents the input matrix, Y presents the output, which size is decided by the window length and feature numbers. After the necessary preprocessing of raw data, sliding windows [Selvin *et al.*, 2017] is applied to the training dataset. The number of generations and individuals are selected. Initial architectures, namely generation 0, are established randomly according to the individual (I) gene encoding as shown in Figure 3. The choice range of each gene position is defined as Equation 6 and Equation 7 given.

For i^{th} generation in G, tournament selection, messy-one-point crossover as shown in Figure 4, and customized integer mutation as shown in Figure 5 as well as their corresponding filters, are applied as GA operations; and generate new generation i^{th+1}. For each individual in I, stacked LSTM model M are established according to their gene encoding L and N. The training set is used to train the weights, while the validation set is used to calculate the MSE.

Hall of Fame [Sivanandam *et al.*, 2008] is considered to save the fittest survivals, and updated after every individual evaluation. All individuals in the list are ranked according to their fitness as shown in Equation 8, and top k performance architectures are printed as well as their fitness at the end of the loop. Each candidate architecture from (A) is evaluated on the testing set m times, and the average MSE E as well as running time T of different architectures, are obtained individually.

5 Numerical Experiments

To evaluate our proposed method, four time-series datasets from different fields with different sampling frequencies are considered in the numerical experiments section. Data cleaning, feature selection, one-hot encoding, normalization, and data splitting are used as potential preprocessing methods, according to their characteristic and quality.

The chosen window size (input length), output length, input features, and output features comprehensively consider the convergence performance and computation burden. Learning rate and θ in Equation 7 are depends on the complexity of forecasting. l in Equation 6 is set as 3, which limitation of architecture depth, while n in Equation 7 is also set as 3, so that the maximum neuron difference among different layers is 8 times, which is complex enough to find representative regularities. An appropriate epoch number is chosen so that the performance difference among different architectures is relatively large. Other details are given in Table 1.

For two layers architecture, we divided the individuals into 7 groups, according to the number of second-layer neurons over the number of first-layer neurons, namely 1/8, 1/4, 1/2, 1, 2, 4, and 8 respectively. For three layers architecture, we divided the individuals into 6 groups, which are AAA, AAB, ABB, ABC, BAB, and BAC, where $A < B < C$.

Some important illustrations are given in Figure 6 to Figure 11. The MSE and running time in which are the average value of 10 times validation calculated by the NVIDIA Tesla M60 GPU. To make the results display concise and clear, we omit some results that are too close to each other in the scatter plot, and guarantee the best results in each group. The complete statistical data for all candidate architectures of different datasets are available in Appendix A.

Figure 6 gives the average MSE and running time of different 2 layers architectures for New South Wales electricity price prediction. According to Figure 6, the MSE almost

[1]https://aemo.com.au/en/energy-systems/electricity/national-electricity-market-nem/data-nem/aggregated-data

[2]https://www.kaggle.com/datasets/camnugent/sandp500

[3]https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume

[4]https://www.kaggle.com/datasets/yasserh/walmart-dataset

[5]Ignore rain, snow, cloud, weather descriptions; one-hot encode weather main and holiday features; normalize temperature value

Table 1: Datesets Details

Datasets	**Electricity Price**	**S&P 500 Price**	**Traffic Volume**	**Walmart Sales**
Source	AEMO [1]	Kaggle [2]	MLR [3]	Kaggle [4]
From Date	2020/1/1	2000/9/18	2012/10/2	2010/2/5
To Date	2022/12/31	2022/12/30	2018/9/30	2012/10/26
Sample Frequency	half hourly	daily	hourly	weekly
Dateset Length	52608	5678	48204	143
Training	40000	4000	35200	143
Validation	10000	1000	8800	143
Testing	2608	678	4204	143
Input Size	48	90	24	52
Output Size	1	**14**	1	1
Input Features	1	5	**25** [5]	45 (grouped in stores)
Output Features	1	1 (close price)	1 (traffic volume)	**45**
Learning Rate	0.001	0.1	0.001	0.0001
Epoch	120	200	20	1000
θ in Equation 7	32	4	64	16

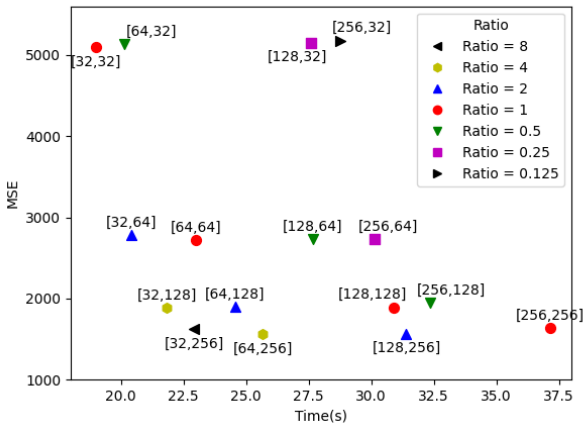

Figure 6: NSW electricity price two layers

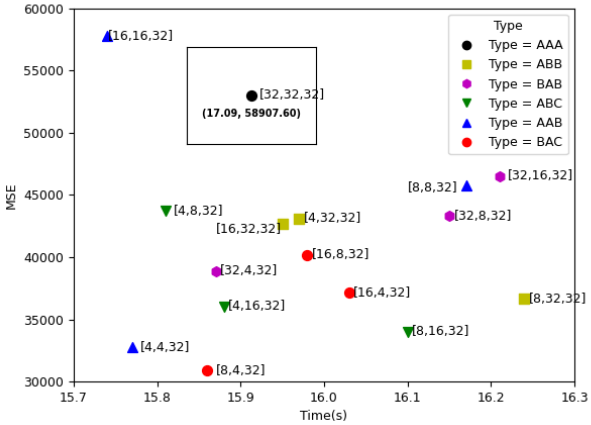

Figure 8: S&P 500 price three layers

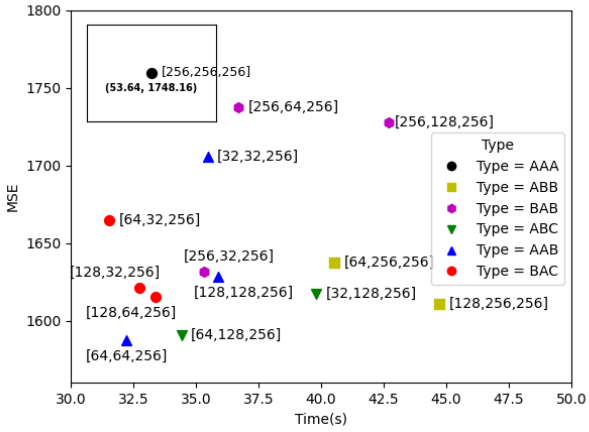

Figure 7: NSW electricity price three layers

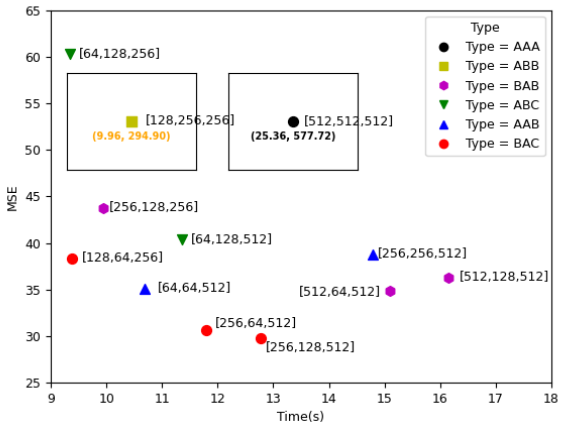

Figure 9: metro interstate traffic volume three layers

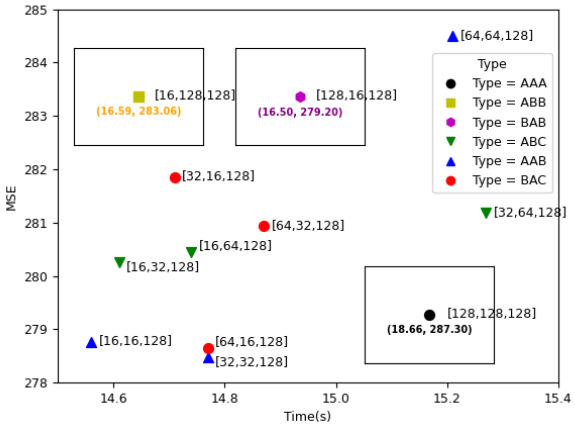

Figure 10: Walmart sales three layers part 1

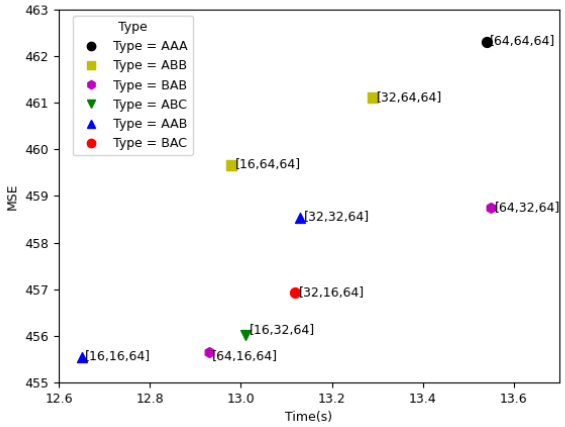

Figure 11: Walmart sales three layers part 2

depends on the number of neurons of the second layer: the larger the neuron, the better the performance. Another result that can be obtained is the larger the ratio of second-layer neuron over first-layer neuron, the better the performance for both shorter running time and smaller MSE value. Similar results are given for the other three datasets.

Figure 7 gives the average MSE and running time of different 3 layers architectures for NSW electricity price prediction. While pattern BAC is better than other patterns from the perspective of running time, pattern ABC has advantages from the perspective of smaller MSE. ABB, AAB, and BAB patterns can only obtain non-optimal solutions according to Pareto optimal solution [Blume *et al.*, 2008]. AAA is the worst choice for slightly larger MSE and is extremely time-consuming, which can be seen in parenthesis of the subplot.

Figure 8 gives the average MSE and running time of different 3 layers architectures for S&P500 price forecasting. The patterns AAB, ABC, and BAC show similar performance, while ABB and BAB can only obtain non-optimal solutions, and AAA given in the subplot continues the longest running time, and the MSE of which is almost doubled compared with optimal architecture.

Figure 9 gives the average MSE and running time of different 3 layers architectures for metro interstate traffic volume prediction. The MSE difference caused by the last layer neurons is not obvious like previously. The patterns ABC and BAC show the best performance, while AAB and BAB consume more running time. AAA and ABB share the worst performance for longer running times. In this scenario [128,64,256] only needs 37.03% running time and [256,128,512] reaches 5.16% MSE accuracy compared with [512,512,512].

Figure 10 and Figure 11 give the average MSE and running time of different 3 layers architectures for Walmart sales forecasting. Figure 10 illustrates the smaller MSE group with 128 neurons on the last layer; Figure 11 illustrates the shorter running time group with 64 neurons on the last layer. According to Figure 10, AAB, ABC, and BAC are still the first class of performance, while ABB and BAB follow with longer running time, and AAA still has the longest running time. According to Figure 11, while the differences among best performance architecture in each group are small, we can conclude that the smaller the neuron numebr of middle layer, the better the performance.

For two-layer stacked LSTM architecture used to do time series forecasting, the MSE almost depends on the number of neurons of the second layer, the larger the neuron, the better the performance. Additionally, the larger the ratio of second-layer neurons over first-layer ones, the better the performance for both shorter running time and smaller MSE value. For three layers stacked LSTM architecture, the dominant affection of the last layer neuron is not obvious like in two layers. The pattern AAB, ABC, and BAC are the first class of performance, while ABB and BAB follow, and AAA always has the longest running time, and sometimes even largest MSE. As the window size and feature numbers for both input and output are variate in different datasets, we have reason to believe that the results obtained are generalizable.

6 Conclusion

We have proposed a GA-based stacked LSTM architecture search algorithm, which extends traditional architecture search space by considering the number of neurons in different layers independently. The traditional crossover and mutation operations in GA have been modified, and the balance between search space and search efficiency has been achieved. The proposed algorithm has been used on four real0world open0source datasets in different fields, and the performance of candidate architectures has been verified.

For two layers stacked LSTM, increasing relation neuron number can achieve similar MSE and less running time compared with the consistent one; while for three layers architectures the running time maximum reduce by 62.97% and MSE reduce at most 19.39 times. The discovered effective architecture patterns may provide clues for the initialization of meta-heuristic architecture search. Additionally, the proposed algorithm can be generalized to other types of neural networks, like GRU and ResNet, or fully connected ones.

References

[Ahmed, 1979] Mohamed Samir Ahmed. *Analysis of freeway traffic time series data and their application to incident detection.* The University of Oklahoma, 1979.

[Ashok *et al.*, 2017] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. *arXiv preprint arXiv:1709.06030*, 2017.

[Bai *et al.*, 2018] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[Bender *et al.*, 2018] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR, 2018.

[Bergstra *et al.*, 2013] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

[Blume *et al.*, 2008] Lawrence Blume, Steven Durlauf, and Lawrence E Blume. *The new Palgrave dictionary of economics*. Palgram Macmillan, 2008.

[Cai *et al.*, 2018] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[Chatfield, 2000] Chris Chatfield. *Time-series forecasting*. CRC press, 2000.

[Cho *et al.*, 2014] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[Conejo *et al.*, 2005] Antonio J Conejo, Miguel A Plazas, Rosa Espinola, and Ana B Molina. Day-ahead electricity price forecasting using the wavelet transform and arima models. *IEEE transactions on power systems*, 20(2):1035–1042, 2005.

[Contreras *et al.*, 2003] Javier Contreras, Rosario Espinola, Francisco J Nogales, and Antonio J Conejo. Arima models to predict next-day electricity prices. *IEEE transactions on power systems*, 18(3):1014–1020, 2003.

[Cui *et al.*, 2020] Zhiyong Cui, Ruimin Ke, Ziyuan Pu, and Yinhai Wang. Stacked bidirectional and unidirectional lstm recurrent neural network for forecasting network-wide traffic state with missing values. *Transportation Research Part C: Emerging Technologies*, 118:102674, 2020.

[Elsken *et al.*, 2019] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[Gauci and Stanley, 2007] Jason Gauci and Kenneth Stanley. Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 997–1004, 2007.

[Gers *et al.*, 2000] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[Goldberg and Sastry, 2007] David Goldberg and Kumara Sastry. *Genetic algorithms: the design of innovation*. Springer, 2007.

[Goodfellow *et al.*, 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[Gorgolis *et al.*, 2019] Nikolaos Gorgolis, Ioannis Hatzilygeroudis, Zoltan Istenes, and Lazlo-Grad Gyenne. Hyperparameter optimization of lstm network models through genetic algorithm. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–4. IEEE, 2019.

[Greff *et al.*, 2016] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[Heule and Kullmann, 2017] Marijn JH Heule and Oliver Kullmann. The science of brute force. *Communications of the ACM*, 60(8):70–79, 2017.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Holland, 1992] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[Jiang *et al.*, 2019] Yufan Jiang, Chi Hu, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. Improved differentiable architecture search for language modeling and named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3585–3590, 2019.

[Liu *et al.*, 2018] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[Miller *et al.*, 1989] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

[Pham *et al.*, 2018] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[Real *et al.*, 2017] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan,

Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

[Ren *et al.*, 2021] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34, 2021.

[Selvin *et al.*, 2017] Sreelekshmy Selvin, R Vinayakumar, EA Gopalakrishnan, Vijay Krishna Menon, and KP Soman. Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 international conference on advances in computing, communications and informatics (icacci)*, pages 1643–1647. IEEE, 2017.

[Shukla *et al.*, 2015] Anupriya Shukla, Hari Mohan Pandey, and Deepti Mehrotra. Comparative review of selection techniques in genetic algorithm. In *2015 international conference on futuristic trends on computational analysis and knowledge management (ABLAZE)*, pages 515–519. IEEE, 2015.

[Sivanandam *et al.*, 2008] SN Sivanandam, SN Deepa, SN Sivanandam, and SN Deepa. *Genetic algorithms*. Springer, 2008.

[Stanley and Miikkulainen, 2002] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[Stanley, 2007] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8:131–162, 2007.

[Sunny *et al.*, 2020] Md Arif Istiake Sunny, Mirza Mohd Shahriar Maswood, and Abdullah G Alharbi. Deep learning-based stock price prediction using lstm and bi-directional lstm model. In *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, pages 87–92. IEEE, 2020.

[Zoph and Le, 2016] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[Zoph *et al.*, 2018] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

A Appendix

Table 2: NSW Electricity Price Two Layers Architecture Search

Ratio	Architecture	MES	TIME
1	[32,32]	5091.4295	19.02
2	[32,64]	2778.0082	20.41
4	[32,128]	1881.5464	21.83
8	[32,256]	1620.4828	22.92
0.5	[64,32]	5133.0862	20.12
1	[64,64]	2721.6668	23.00
2	[64,128]	1893.7312	24.56
4	[64,256]	1556.5624	25.68
0.25	[128,32]	5151.1624	27.60
0.5	[128,64]	2734.3004	27.66
1	[128,128]	1890.1470	30.90
2	[128,256]	1561.3165	31.37
0.125	[256,32]	5176.7471	28.78
0.25	[256,64]	2729.8328	30.13
0.5	[256,128]	1944.0744	32.35
1	[256,256]	1635.9198	37.14

Table 3: NSW Electricity Price Three Layers Architecture Search

Type	Architecture	MES	TIME
AAA	[32,32,32]	5056.0832	30.10
AAB	[32,32,64]	2692.1960	33.02
AAB	[32,32,128]	1912.5928	34.45
AAB	[32,32,256]	1705.7308	35.48
ABB	[32,64,64]	2737.0400	36.73
ABC	[32,64,128]	1876.6614	37.26
ABC	[32,64,256]	1622.1298	37.87
ABB	[32,128,128]	3001.2496	39.37
ABC	[32,128,256]	1617.1493	39.81
ABB	[32,256,256]	2061.5221	39.69
BAB	[64,32,64]	2695.9832	30.61
BAC	[64,32,128]	1919.3351	30.62
BAC	[64,32,256]	1664.9756	31.55
AAA	[64,64,64]	2701.0581	32.01
AAB	[64,64,128]	1849.3826	33.51
AAB	[64,64,256]	1587.3296	32.23
ABB	[64,128,128]	1812.9305	34.73
ABC	[64,128,256]	1590.8074	34.42
ABB	[64,256,256]	1637.4133	40.53
BAB	[128,32,128]	1914.8700	33.30
BAC	[128,32,256]	1621.2188	32.75
BAB	[128,64,128]	1966.6744	34.68
BAC	[128,64,256]	1615.1799	33.40
AAA	[128,128,128]	1893.2439	34.01
AAB	[128,128,256]	1628.1431	35.90
ABB	[128,256,256]	1611.0903	44.71
BAB	[256,32,256]	1631.6770	35.33
BAB	[256,64,256]	1737.5995	36.69
BAB	[256,128,256]	1727.5942	42.71
AAA	[256,256,256]	1748.1610	55.64

Table 4: S&P500 Two Layers Architecture Search

Ratio	Architecture	MES	TIME
1	[4,4]	494000.7875	10.64
2	[4,8]	211372.5656	10.6
4	[4,16]	77421.9453	10.61
8	[4,32]	37995.5359	10.65
0.5	[8,4]	516715.5375	10.67
1	[8,8]	245843.3703	10.72
2	[8,16]	74695.6906	10.71
4	[8,32]	39101.8768	10.67
0.25	[16,4]	492932.3906	10.65
0.5	[16,8]	261801.1938	10.73
1	[16,16]	133329.7531	10.65
2	[16,32]	48509.327	10.61
0.125	[32,4]	579379.35	10.72
0.25	[32,8]	359292.6625	10.73
0.5	[32,16]	189563.4641	10.71
1	[32,32]	40647.1379	10.91

Table 5: S&P500 Three Layers Architecture Search

Type	Architecture	MES	TIME
AAA	[4,4,4]	227631.5250	15.47
AAB	[4,4,8]	222729.9891	15.56
AAB	[4,4,16]	120300.3383	15.68
AAB	[4,4,32]	32739.2875	15.77
ABB	[4,8,8]	248856.8594	15.44
ABC	[4,8,16]	115505.8852	15.67
ABC	[4,8,32]	43729.7699	15.81
ABB	[4,16,16]	95377.9309	15.80
ABC	[4,16,32]	36037.7512	15.88
ABB	[4,32,32]	43085.3986	15.97
BAB	[8,4,8]	227636.7047	15.62
BAC	[8,4,16]	88354.6109	15.74
BAC	[8,4,32]	30873.9998	15.86
AAA	[8,8,8]	249751.3672	15.83
AAB	[8,8,16]	88322.2379	16.05
AAB	[8,8,32]	45794.7607	16.17
ABB	[8,16,16]	151799.1203	16.03
ABC	[8,16,32]	34010.9867	16.10
ABB	[8,32,32]	36657.8934	16.24
BAB	[16,4,16]	87277.9770	15.88
BAC	[16,4,32]	37148.6551	16.03
BAB	[16,8,16]	120627.6078	15.87
BAC	[16,8,32]	40200.8906	15.98
AAA	[16,16,16]	182439.1016	15.47
AAB	[16,16,32]	57737.2201	15.74
ABB	[16,32,32]	42701.1834	15.95
BAB	[32,4,32]	38904.1967	15.87
BAB	[32,8,32]	43298.6152	16.15
BAB	[32,16,32]	46529.3387	16.21
BAB	[32,32,32]	58907.5959	17.09

Table 6: Metro Interstate Traffic Volume Two Layers Architecture Search

Ratio	Architecture	MES	TIME
1	[64,64]	400.9534	6.33
2	[64,128]	85.7968	6.47
4	[64,256]	39.3684	6.68
8	[64,512]	34.6527	8.74
0.5	[128,64]	424.5357	6.79
1	[128,128]	82.7868	6.72
2	[128,256]	34.2314	6.94
4	[128,512]	29.5516	9.38
0.25	[256,64]	474.9874	7.02
0.5	[256,128]	87.3822	7.14
1	[256,256]	32.9244	7.71
2	[256,512]	27.1124	11.19
0.125	[512,64]	666.4767	9.11
0.25	[512,128]	99.4253	9.63
0.5	[512,256]	33.1705	10.99
1	[512,512]	28.0687	16.15

Table 7: Metro Interstate Traffic Volume Three Layers Architecture Search

Type	Architecture	MES	TIME
AAA	[512,512,512]	577.7212	25.36
AAB	[64,64,512]	35.1290	10.69
AAB	[128,128,512]	32.1118	11.87
AAB	[256,256,512]	38.7726	14.78
ABC	[64,128,512]	40.3817	11.35
BAB	[512,64,512]	34.9069	15.09
BAB	[512,128,512]	36.3124	16.15
BAC	[128,64,512]	32.7873	11.27
BAC	[256,64,512]	30.6024	11.80
BAC	[256,128,512]	29.7954	12.77
AAB	[64,64,256]	42.9847	9.56
AAB	[128,128,256]	41.1886	9.99
ABC	[64,128,256]	60.2497	9.34
BAC	[128,64,256]	38.3396	9.39
BAB	[256,64,256]	38.9563	9.43
BAB	[256,128,256]	43.7931	9.95
AAA	[256,256,256]	132.3078	11.25
ABC	[128,256,512]	168.3304	13.62
BAB	[512,256,512]	151.1722	18.26
ABB	[128,256,256]	294.8985	9.96
ABB	[64,512,512]	717.5033	17.34
ABB	[128,512,512]	679.2443	17.46
ABB	[256,512,512]	601.5514	19.15
ABC	[64,256,512]	565.4804	12.85
ABB	[64,256,256]	707.1509	9.89
AAB	[64,64,128]	196.5013	9.33
ABB	[64,128,128]	717.1946	9.19
BAB	[128,64,128]	179.6921	9.72
AAA	[128,128,128]	379.8769	9.85
AAA	[64,64,64]	750.0319	8.98

Table 8: Walmart Sales Two Layers Architecture Search

Ratio	Architecture	MES	TIME
1	[16,16]	680.7387	9.41
2	[16,32]	592.6301	9.39
4	[16,64]	456.782	9.36
8	[16,128]	279.9863	11.35
0.5	[32,16]	677.9800	9.41
1	[32,32]	592.2019	9.33
2	[32,64]	459.6788	9.82
4	[32,128]	282.5249	11.22
0.25	[64,16]	677.1815	9.63
0.5	[64,32]	594.3028	9.94
1	[64,64]	460.449	9.72
2	[64,128]	281.1379	11.62
0.125	[128,16]	680.8431	11.34
0.25	[128,32]	596.6989	12.35
0.5	[128,64]	465.2175	12.36
1	[128,128]	285.8920	13.96

Table 9: Walmart Sales Three Layers Architecture Search

Type	Architecture	MES	TIME
AAA	[16,16,16]	673.0845	12.85
AAB	[16,16,32]	589.5641	13.01
AAB	[16,16,64]	455.5526	12.65
AAB	[16,16,128]	278.77	14.56
ABB	[16,32,32]	591.5358	12.77
ABC	[16,32,64]	456.0120	13.01
ABC	[16,32,128]	280.2498	14.61
ABB	[16,64,64]	459.6646	12.98
ABC	[16,64,128]	280.4427	14.74
ABB	[16,128,128]	283.065	16.59
BAB	[32,16,32]	588.1675	12.79
BAC	[32,16,64]	456.9272	13.12
BAC	[32,16,128]	281.8531	14.71
AAA	[32,32,32]	590.7461	12.97
AAB	[32,32,64]	458.5329	13.13
AAB	[32,32,128]	278.481	14.77
ABB	[32,64,64]	461.1048	13.29
ABC	[32,64,128]	281.1766	15.27
ABB	[32,128,128]	283.876	16.37
BAB	[64,16,64]	455.6574	12.93
BAC	[64,16,128]	278.6441	14.77
BAB	[64,32,64]	458.7436	13.55
BAC	[64,32,128]	280.9329	14.87
AAA	[64,64,64]	462.3017	13.54
AAB	[64,64,128]	284.4942	15.21
ABB	[64,128,128]	284.0521	16.84
BAB	[128,16,128]	279.2043	16.5
BAB	[128,32,128]	278.971	17.78
BAB	[128,64,128]	282.424	17.38
AAA	[128,128,128]	287.3026	18.66